

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-010943

(43)Date of publication of application : 14.01.2000

(51)Int.CI.

G06F 15/16

(21)Application number : 11-140498

(71)Applicant : INTERNATL BUSINESS MACH CORP <IBM>

(22)Date of filing : 20.05.1999

(72)Inventor : THOMAS YUKYU QUACK CHANDORASEKUHAARU NARAYANASUWAMI BENT-OREIF SCHNEIDER

(30)Priority

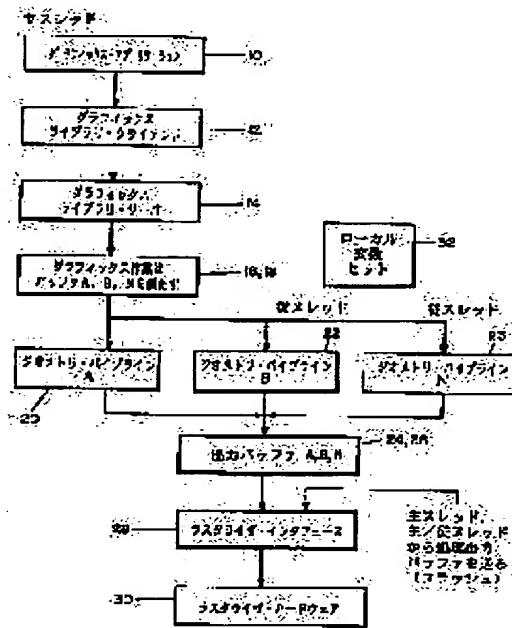
Priority number : 98 87093 Priority date : 29.05.1998 Priority country : US

(54) METHOD AND SYSTEM FOR PROCESSING INPUT DATA THROUGH DATA PROCESSOR PIPELINE

(57)Abstract:

PROBLEM TO BE SOLVED: To make it possible to use plural threads for the graphics pipeline of an image rendering system by using local variables which are accessible to both a master thread and a slave thread for a step wherein an input buffer is allocated and a step wherein it is decided whether or not the input buffer can be allocated.

SOLUTION: When there is no slave thread, a slave thread is generated and the input buffer is allocated to the slave thread for processing. When a slave thread is already present, the input buffer is allocated to the slave thread and when the input buffer can not be allocated, the input buffer is allocated to the main thread for processing which is performed in parallel to a process performed by the slave thread. The local variables which are accessible to both the main thread and slave thread are used for the step for allocating the input buffer and the step for deciding whether the allocation is possible. In a graphics task, a local variable set 32 stores variable flags, which can be set and reset by the main thread or slave thread.



BEST AVAILABLE COPY

LEGAL STATUS

[Date of request for examination] 17.09.1999

[Date of sending the examiner's decision of rejection] 23.07.2002

[Kind of final disposal of application other than the examiner's decision of rejection or

[application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19)日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開2000-10943

(P2000-10943A)

(43)公開日 平成12年1月14日(2000.1.14)

(51) Int.Cl.⁷

識別記号

F I
G.06.E 15/16

テーマコード（参考）

G 0 6 F 15/16

610

G 0 6 F 15/16

610 F

審査請求 有 請求項の数14 OL (全 19 頁)

(21) 出願番号 特願平11-140498

(22)出願日 平成11年5月20日(1999.5.20)

(31) 優先権主張番号 09/087093

(32)優先日 平成10年5月29日(1998.5.29)

(33) 優先権主張国 米国 (U.S.)

(71) 出願人 390009531

インターナショナル・ビジネス・マシーンズ・コーポレイション
INTERNATIONAL BUSINESS MACHINES CORPORATION
アメリカ合衆国10504、ニューヨーク州
アーモンク（番地なし）

(74) 代理人 100086243

弁理士 坂口 博 (外1名)

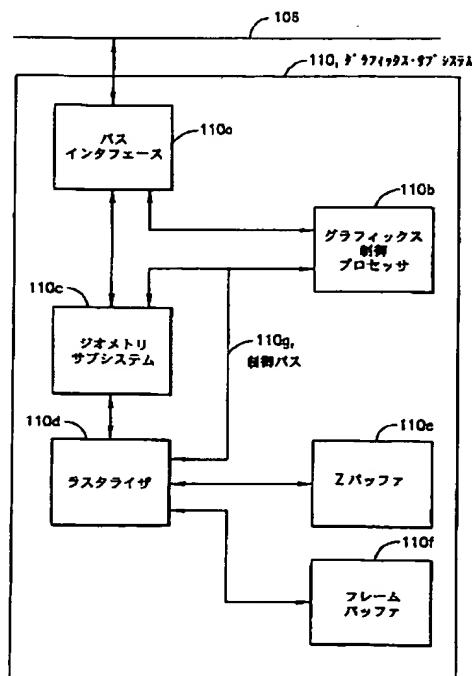
最終頁に統く

(54) 【異常の名称】 インカーデータをデータプロセッサ・パイプラインで処理する方法およびシステム

(57) 【要約】

【課題】 マルチプロセッサ・システムにおいて、入力データをデータプロセッサ・パイプラインで処理する方法およびシステムを提供する。

【解決手段】 この方法は、主スレッドを操作して、入力バッファがフルになるまで、あるいは入力データが終了するまで、入力バッファに入力データを格納するステップを含んでいる。従スレッドが存在しない場合には、従スレッドを生成して、入力バッファを、処理のために従スレッドに割当てる。従スレッドが既に存在する場合には、従スレッドを入力バッファに割当てることができるか否かを判別し、割当てることができる場合には、入力バッファを、処理のために従スレッドに割当てる。従スレッドを入力バッファに割当てることができない場合には、主スレッドが、入力バッファを従スレッドによって実行される処理との並列処理のために、自身に割当てる。



【特許請求の範囲】

【請求項1】マルチプロセッサ・システムにおいて、入力データをデータプロセッサ・パイプラインで処理する方法であって、
主スレッドを操作して、入力バッファがフルになるまで、あるいは入力データが終了するまで、前記入力バッファに前記入力データを格納するステップと、
従スレッドが存在しない場合には、従スレッドを生成して、前記入力バッファを、処理のために従スレッドに割当てるステップと、
従スレッドが既に存在する場合には、前記従スレッドを前記入力バッファに割当てることができるか否かを判別し、割当てることができる場合には、前記入力バッファを、処理のために従スレッドに割当てる、割当てることができない場合には、前記入力バッファを、前記従スレッドによって実行される処理との並列処理のために、前記主スレッドに割当てるステップとを含み、
前記割当てるステップおよび判別するステップは、前記主スレッドおよび従スレッドの両方にアクセス可能なローカル変数を用いる、ことを特徴とする方法。

【請求項2】処理データバッファを前記パイプラインの連続する処理ステージに送るステップをさらに含み、このステップは、前記データバッファが、前記従スレッドによってまたは前記主スレッドによって処理されたかにかかわらず、前記従スレッドによってのみ実行されることを特徴とする請求項1記載の方法。

【請求項3】処理データバッファを前記パイプラインの連続する処理ステージに送るステップをさらに含み、このステップは、前記データバッファが、前記従スレッドによって、他の従スレッドによって、または前記主スレッドによって処理されたかにかかわらず、前記従スレッドによってのみ実行されることを特徴とする請求項1記載の方法。

【請求項4】前記入力データは、ジオメトリ・イメージをレンダリングするための、頂点座標の記述であることを特徴とする請求項1記載の方法。

【請求項5】マルチプロセッサ・システムにおいて、主スレッドの操作と、少なくとも1つの従スレッドの操作とを同期させる方法であって、

主スレッドを操作して、入力バッファがフルになるまで、あるいは入力データが終了するまで、前記入力バッファに前記入力データを格納するステップと、

従スレッドが存在しない場合には、従スレッドを生成して、前記入力バッファを、処理のために従スレッドに割当てるステップと、

従スレッドが既に存在する場合には、前記従スレッドを前記入力バッファに割当てることができるか否かを判別し、割当てることができる場合には、前記入力バッファを、処理のために従スレッドに割当てる、割当てることができない場合には、前記入力バッファを、前記従スレッ

ドによって実行される処理との並列処理のために、前記主スレッドに割当てるステップとを含み、
前記割当てるステップおよび判別するステップは、オペレーティング・システム・コールの使用を要求することなく、前記主スレッドおよび従スレッドの両方にアクセス可能なローカル変数を用いて同期される、ことを特徴とする方法。

【請求項6】処理データバッファを前記パイプラインの連続する処理ステージに送るステップをさらに含み、このステップは、前記データバッファが、前記従スレッドによってまたは前記主スレッドによって処理されたかにかかわらず、前記ローカル変数を用いて、前記従スレッドによってのみ実行されることを特徴とする請求項5記載の方法。

【請求項7】処理データバッファを前記パイプラインの連続する処理ステージに送るステップをさらに含み、このステップは、前記データバッファが、前記従スレッドによって、他の従スレッドによって、または前記主スレッドによって処理されたかにかかわらず、前記ローカル変数を用いて、前記従スレッドによってのみ実行されることを特徴とする請求項5記載の方法。

【請求項8】前記入力データは、ジオメトリ・イメージをレンダリングするための、頂点座標の記述であることを特徴とする請求項5記載の方法。

【請求項9】グラフィックス・ジオメトリ・パイプラインにおいて、表示のためにレンダリングされるグラフィカル・モデルを記述する頂点座標のデータストリームを処理する方法であって、

主スレッドを操作して、前記グラフィカル・モデルの第1の状態に関連した頂点座標データストリームの開始を検出するステップと、

第1のデータ入力バッファがフルになるまで、あるいは前記入力頂点座標データストリームが終了するまで、前記第1のデータ入力バッファに前記頂点座標データストリームを格納するステップと、

第1の従スレッドが存在するか否かを判別するステップと、

前記第1の従スレッドが存在しない場合には、第1の従スレッドを生成して、前記第1のデータ入力バッファを、処理のために前記第1の従スレッドに割当てるステップと、

前記第1の従スレッドが既に存在する場合には、前記第1の従スレッドを前記第1のデータ入力バッファに割当てることができるか否かを判別し、割当てることができる場合には、前記第1のデータ入力バッファを、処理のために前記第1の従スレッドに割当てるステップと、

第2のデータ入力バッファがフルになるまで、または前記入力頂点座標データストリームが終了するまで、前記第2のデータ入力バッファに頂点座標データストリームをさらに格納するステップと、

前記第1の従スレッドを前記第1のデータ入力バッファに割当てることができないことが判別される場合には、前記主スレッドを操作して、以下の(A)または(B)のステップを実行するステップを含み、

(A) 第2の従スレッドが存在するか否かを判別し、前記第2の従スレッドが存在しない場合には、第2の従スレッドを生成して、前記第2のデータ入力バッファを、処理のために前記第2の従スレッドに割当て、前記第2の従スレッドが既に存在する場合には、前記第2の従スレッドを前記第2のデータ入力バッファに割当てができるか否かを判別し、割当てができる場合には、前記第2のデータ入力バッファを、処理のために前記第2の従スレッドに割当て、

(B) 前記第1および第2の従スレッドの少なくとも1つによって実行される処理と並列に処理するために、前記第2のデータ入力バッファを前記主スレッドに割当て、

前記割当てのステップおよび前記判別のステップは、オペレーティング・システム・コールの使用を要求することなく、前記主スレッドと前記第1および第2の従スレッドとを両方にアクセス可能なローカル変数を用いて同期される、ことを特徴とする方法。

【請求項10】処理データバッファを、前記グラフィックス・ジオメトリ・パイプラインの連続するステージに送るステップをさらに含み、このステップは、前記データバッファが、前記第1の従スレッドによって、前記第2の従スレッドによって、または前記主スレッドによって処理されたかにかかわらず、前記ローカル変数を用いて、前記第1の従スレッドによってのみ実行されることを特徴とする請求項9記載の方法。

【請求項11】前記連続するステージは、ラスタライザ・ステージであることを特徴とする請求項10記載の方法。

【請求項12】前記主スレッドを操作して、前記グラフィカル・モデルの第2の状態の発生を検出するステップと、

前記グラフィカル・モデルの第1の状態に関連した頂点座標データのすべての処理を終了するステップと、をさらに含むことを特徴とする請求項10記載の方法。

【請求項13】複数のデータプロセッサを備え、各データプロセッサは、主スレッドを実行し、少なくとも1つの第2のデータプロセッサは、従スレッドを実行する、グラフィックス・データ処理システムであって、

前記主スレッドと前記少なくとも1つの従スレッドとの両方によってアクセス可能な1組のローカル変数を格納するメモリ手段と、

前記主スレッドの制御のもとで、入力グラフィックス・データストリームを格納する複数の入力バッファと、前記ローカル変数に応答する前記主スレッドに関連した処理手段であって、入力バッファを、前記主スレッドと

前記従スレッドとの間に割当て、前記入力バッファに格納されたデータを用いて、グラフィックス・データ演算タスクを並列に実行するために、前記主スレッドの操作と従スレッドの操作とを同期させる処理手段と、を備えることを特徴とするグラフィックス・データ処理システム。

【請求項14】前記メモリ手段は、前記第1のデータプロセッサに接続された第1のキャッシュメモリと、前記第2のデータプロセッサに接続された第2のキャッシュメモリとよりなり、前記第1および第2のキャッシュメモリの各々は、前記1組のローカル変数の同一コピーを格納することを特徴とする請求項13記載のグラフィックス・データ処理システム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、一般的には、複数のデータプロセッサ（すなわち、マイクロプロセッサ・システム）を有するデータ処理システムに関し、特に、マルチプロセッサ・システムにおいて作業割当てを並列化し、および配布する方法に関する。

【0002】

【従来の技術】図1において、マルチプロセッサ1は、2個以上のデータプロセッサ（例えばP0～P3）を有するマシンである。データプロセッサは、バスまたはクロスバースイッチ2によって互いに接続することができる。各プロセッサは、関連したキャッシュメモリ（C0～C3）を有することができる。プロセッサP0～P3は、バスまたはクロスバースイッチ2と、関連するキャッシュメモリ（設けられているならば）を介して、共通のシステムメモリ3を共有している。各プロセッサは、他のプロセッサにアクセスできないプライベートメモリ（PM）を有することもできる。

【0003】マルチプロセッサ1の各プロセッサP0～P3は、関連したタスクを実行することができる。例えば、オーディオ・アプリケーションまたはタスクを、1つのプロセッサで実行でき、他方、ビデオ・アプリケーションを他のプロセッサで実行することができる。この場合、各プロセッサは、他のプロセッサで実行されているタスクの間になんらかの強い相互作用を生じることなく、実質的に独立して、そのタスクを実行する。

【0004】本発明について最も興味ある他の場合において、1つのタスクは、サブタスクに区分され、サブタスクは、1つのプロセッサを1つのサブタスクに割当てることによって、2つ以上のプロセッサ上で共働して実行される。いくつかのプロセッサが、このように共働して、1つのタスクを実行する場合には、いくつかのプロセッサは、メモリ3、バッファ、プリンタ、他の周辺機器（図示せず）のような共通リソースを公平に共用することを典型的に必要とする。さらにプロセッサは、互いに通信して、チェックポイントで必要とされる情報を共

用し、他のプロセッサが一定のルーチンを終了するのを待ち、他のプロセッサにプロセッサがその割当てられたサブタスクを終了したことを知らせることなどを典型的に必要とする。

【0005】“スレッド (thread)”は、いくつかのタスクを1つのプロセスで作成できる環境におけるプロセスの機能である。特に、スレッドは、1つのアドレス空間を共用する1組のサブプロセッサのうちの1つである。この場合、オフ・スタック (グローバル) 変数が、プログラムのすべてのスレッドの間で共用される。各スレッドは、自身の個別のローカル変数を有する個別のコール・スタックを実行する。プロセス内のすべてのスレッドは、プロセスID、プロセスグループID、セッションメンバーシップ、有効な保存されたセットユーザID、有効な保存された実セットグループID、サブリメンタリグループID、現行の作業ディレクトリ、ルートディレクトリ、ファイルモード作成マスク、ファイル記述子のようなシステムリソースを共用する。システムリソースの前述したリストは、一例であり、これらリソースのすべてを、アプリケーションで用いることはできず、あるいはリストされたこれらのリソースより多くのリソースを用いることができる。サブプロセス・グループの唯一のメンバーであるスレッドは、プロセスに相当している。

【0006】カーネル・スレッドは、カーネル空間内のスレッドの実行に関係している。カーネル空間は、ユーザ・アプリケーションにアクセスできない特権空間であると、技術上典型的にみなされている。ユーザ・スレッドは、ユーザ空間内でのスレッドの実行に関係している。スレッド環境では、m個のユーザ・スレッドを、n個のカーネル・スレッドにマップすることができる。

【0007】スレッドセーフ・ライブラリは、スレッドセーフ機能を含むライブラリである。スレッドセーフ機能は、複数のスレッドによって共働的に安全に呼出うことのできる機能である。スレッドセーフ環境における再入可能機能は、次のような機能である。すなわち、2つ以上のスレッドによって呼出されるときに、機能の効果が、実際の実行がたとえインタリープされても、あたかも機能が不定の順序で1つずつ実行されるように、保証される機能である。ライブラリ機能は、スレッドセーフとみなされるライブラリに対して、再入可能でなければならない。

【0008】現在利用できるスレッド・ソフトウェアパッケージは、典型的に、スレッドを作成し、ある機能の実行を開始する機能を有している。新しく作成されたスレッドは、それが実行する機能が終了したとき、あるいはスレッドが明らかに終わったときに、終了する。スレッド・パッケージは、また典型的に、ミューテックス (mutex)，条件変数、セマフォ (semaphore) のような種々の同期プリミティブを与え、他の

スレッドから送付されるイベントを待ち、イベントを他のスレッドに送付するなどを行う。これらのスレッド関連コンセプトの特定の詳細は、刊行物 “Operating Systems Principles”, Prentice Hall 1973, Per Brinch Hansen著、または “Cooperative Sequential Processes”, Technical Report Technological University 1965, E. W. Dijkstra著から得られる。

【0009】スレッドを作成しおよび破壊することは、プロセスを作成し破壊することよりも計算的には高価ではないが、わずかな作業またはタスクが並列に実行され、高度の同期および通信を必要とする微細粒度で、スレッドを作成しおよび破壊することは、依然として有効ではないことに留意すべきである。

【0010】同期操作は、2つ以上のスレッドがリソースを共用しなければならないときに、含まれる。例えば、スレッドAが、スレッドBによって処理される作業バッファに、作業項目を挿入することであると仮定する。作業項目を挿入した後に、スレッドAは、バッファにおける作業項目のカウントをインクリメントする。同様に、作業項目を処理した後に、スレッドBは、バッファにおける作業項目のカウントをデクリメントする。この例では、バッファは100個の作業項目を保持でき、カウンタは現在58であるものと仮定する。さらに、スレッドAは、カウントを58から59にインクリメントし始め、同時に、スレッドBは、カウントを58から57にデクリメントし始めるものと仮定する。スレッドBが、ディクリメント動作を後に終了すると、カウンタは57であり、スレッドAが、インクリメント動作を後に終了すると、カウンタは59である。正しいカウンタ値は58であるので、いずれのカウンタ値も正しくない。この問題は、スレッドAおよびスレッドBの両方が、カウンタ上で同時に動作することが許容されるので発生する。これは、技術上、同期問題と呼ばれている。この問題の解決は、スレッドAがカウンタを変更するときに、スレッドBがカウンタを変更することを許容しないことであり、およびスレッドAがカウンタを変更するときに、スレッドBがカウンタを変更することを許容しないことである。この問題に対する従来の解決手段は、オペレーティング・システムによって与えられる相互排他プリミティブの使用を復活させている。この方法に対する1つの欠点は、実行すべき数十のプロセッサ・サイクルを必要とするシステム・コール・オペレーションを含むことである。その結果、相互排他プリミティブの使用は、作業項目が小さいときには、適切でない。というのは、相互排他プリミティブを用いることのオーバヘッドは、作業を行うために2つのスレッドを用いることによって得ることのできる利点を否定するからである。

【0011】図2は、タスクを並列に実行するアプリケーションの一例の全体構成を概念的に示す。このアプリケーションでは、主スレッドおよび従スレッドは、必要な作業を共働して実行する。主スレッドは、アプリケーションから作業を集めて、それを作業バッファ（タスクバッファA～C）に格納し、従スレッドは、作業バッファに格納された作業項目を実行する。すべての作業バッファが満たされて、バッファが利用できなくなると、作業バッファを選択し、選択された作業バッファにおいて作業項目を実行することによって、主スレッドが、従スレッドを補助する。この方法は、システムにおけるすべてのプロセッサが、最大の効率で利用されることを保証する。というのは、主スレッドが割当てられるプロセッサは、作業バッファが利用できるようになるまでアイドルすることは要求されないからである。主スレッドおよび従スレッドは、作業バッファを同時にアクセスすることを試みることができるので、同期を必要とする状況が生じる。すなわち、ある機構を設けて、主スレッドおよび従スレッドの両方によってではなく、主スレッドまたは従スレッドのいずれかによって、各作業バッファが1回だけ処理されるようにしなければならない。さらに、作業は、有限の時間内に行われるようになることが重要である。すなわち、主スレッドが、従スレッドが作業バッファを処理することを仮定する、あるいは従スレッドが、主スレッドが作業バッファを処理することを仮定する状況は、存在しない。というのは、このような状況が発生すると、作業バッファにおいて作業項目を決して処理させることができないからである。

【0012】従来は、スレッド・ライブラリに与えられる同期プリミティブを用いることによって、同期が実現されている。スレッド・ライブラリの一例は、技術上、POSIX Pthreadsライブラリ（IEEE Standards Project: Draft Standard for Information Technology – Portable Operating System Interface (POSIX) Amendment 2: Threads Extension [C Language] Tech Report P1003.4a Draft 7, IEEE Standards Department, April 23, 1993参照）として知られている。

【0013】リソースを要求する前に、スレッドは、典型的にまず最初に、リソース上のロックを得なければならない。定義によれば、ロックを得るときには、スレッドは、他のスレッドはリソースのためのロックを所有しないこと、およびスレッドはリソースを用いるには自由であることを知っている。第2のスレッドが、リソースを要求することを望むならば、第2のスレッドは、第1のスレッドがリソースを用いて終了するまで、ロックを

得るのは待たなければならない。リソースを用いて第1のスレッドが終了すると、第1のスレッドは、リソースのためのロックを解放し、これにより他のスレッドがリソースをアクセスすることを可能にする。

【0014】この方法を用いることの1つの欠点は、スレッド・ライブラリで定義される、典型的に低速のロック機能を実行しなければならないことである。さらに、実際の実施では、ロック機能の実行は、オペレーティング・システムのサービス（非常に低速のプロセスである）が望まれることを要求する。このような時間ペナルティは、クリティカル・リソースで実行される作業が、非常に時間を消費しない場合に、増大する。したがって、微細粒度の同期の使用を必要とするアプリケーションについて、スレッド・ライブラリを有する同期プリミティブを使用することは、典型的にコスト効率的でない。

【0015】本発明の目的は、前述したおよびその他の同期に関連した問題に対する解決を与え、および微細粒度の同期アプリケーションに用いられる改良された同期方法を提供することにある。

【0016】本発明の他の目的は、画像レンダリング・システムのグラフィックス・パイプラインに、複数のスレッドを用いることを可能にする同期装置および方法を提供することにある。

【0017】

【課題を解決するための手段】前述の問題およびその他の問題は、本発明の実施例による方法および装置によって克服され、および本発明の目的は達成される。

【0018】グラフィックス・データ処理システムのようなマルチプロセッサ・システムにおいて、入力データをデータプロセッサ・パイplineで処理する方法が開示される。この方法は、主スレッドを操作して、入力バッファがフルになるまで、あるいは入力データが終了するまで、入力バッファに入力データを格納するステップを含んでいる。従スレッドが存在しない場合には、従スレッドを生成して、入力バッファを、処理のために従スレッドに割当てる。従スレッドが既に存在する場合には、従スレッドを入力バッファに割当てることができるか否かを判別し、割当てることができる場合には、入力バッファを、処理のために従スレッドに割当てる。従スレッドを入力バッファに割当てることができない場合には、主スレッドが、入力バッファを従スレッドによって実行される処理との並列処理のために、自身に割当てる。2つ以上の従スレッドを生成し操作して、ジオメトリ・サブシステムの高度の並列化を与えることができる。割当てるステップおよび判別するステップは、主スレッドおよび従スレッドの両方にアクセス可能であり、変更および/またはテストにオペレーティング・システム・コールを必要としない、ローカル変数のみを用いることを必要とする。

【0019】この方法は、ラスタライザ・ユニットのような前記バイオラインの連続する処理ステージに、処理データバッファを送るステップをさらに含んでいる。本発明の好適な態様によれば、処理データバッファを送るステップは、データバッファが、従スレッドによってまたは主スレッドによって処理されたかにかかわらず、従スレッドによってのみ実行される。

【0020】

【発明の実施の形態】まず初めに、グラフィックス・バイオライン、および表示ライオラリのラスタライザの要素に、複数のスレッドを用いることについて考察する。複数のスレッドの使用に關し、グラフィックス・プロセスの構成を考察し、並列の全体モデルを使用する。

【0021】本発明を実施するに適した例示的なグラフィックス処理システム100の全体のアーキテクチャを、図3に示す。図示のように、グラフィックス処理システム100は、システム制御プロセッサ102を備え、このプロセッサは、システムバス106を介してシステムメモリ104に接続されている。システムメモリ104は、ランダムアクセスメモリ(RAM)を有し、このRAMは、1つ以上の3次元のモデル/ビューに含まれるオブジェクトを定めるグラフィックス・データを格納する。システムメモリ104は、また、システム制御プロセッサ102で実行されるアプリケーション・プログラムを格納する。このアプリケーション・プログラムは、ユーザ・インターフェースを与えて、ナビゲートスルーリーし、および/またはメモリ104に格納されたグラフィックス・データによって定められる3次元モデル/ビューを変更する。各オブジェクトを定めるグラフィックス・データは、プリミティブの座標および属性(例えばカラー)を含んでいる。プリミティブは、立体、線、面のようなジオメトリック・エンティティである。典型的には、プリミティブは、3つの頂点によって定まる三角形である。この場合、システムメモリ104は、3次元ビューを形成するオブジェクトの面を定める三角形の頂点の番号付きリストを含んでいる。さらに、システムメモリ104は、プリミティブの各々に相当するプリミティブ識別子のリストと、どのように且つどこにプリミティブが表示されるべきかを特定する変換マトリックスとを格納することができる。入力/出力(I/O)装置108は、システムバス106を介して、システム制御プロセッサ102とインターフェースする。I/O装置108は、テキスト入力用のキーボード、テンプレート、タッチパッドのうちの1つ以上と、ユーザ入力用のマウス、トラックボール、ライトペンのようなポインティング・デバイスと、音声入力用の音声認識装置とを有することができる。

【0022】グラフィックス処理システム100は、また、システムバス106を介してシステムメモリ104とインターフェースするグラフィックス・サブシステム1

10を有している。このグラフィックス・サブシステム110は、本発明の教示に対して最も興味のあるものであり、図4に詳細に示されている。一般に、グラフィックス・サブシステム110は、アプリケーション・プログラムからのコマンドのもとで動作し、システムメモリ104に格納されているグラフィックス・データをレンダリングし、表示装置112の表示領域における画素アレイとして表示する。表示装置112は、画素を表示するため、ラスタスキャン技術または液晶表示技術を用いることができる。グラフィックス・サブシステム110によって生成された画素データは、デジタル形式であり、および典型的に、表示装置112は、画素データをアナログ形式で必要とする。この場合、デジタル-アナログ変換器(DAC)114を、グラフィックス・サブシステム110と表示装置112との間に設けて、画素データをデジタルから、表示装置112を駆動するに適したアナログ形式に変換する。

【0023】図4において、グラフィックス・サブシステム110は、グラフィックス・サブシステム110の動作を指示する制御ユニットすなわちグラフィックス制御プロセッサ110bを有している。シーンをレンダリングするグラフィックス・オーダーを受取ると、制御ユニットすなわちグラフィックス制御プロセッサ110bは、グラフィックス・オーダーに関連したグラフィックス・データを、レンダリング・エンジンすなわちジオメトリ・サブシステム110cに送る。レンダリング・エンジン110cは、グラフィックス・オーダーに関連したグラフィックス・データを、モデル座標系からビュー座標系に変換し、所定のビューポリュームに対してグラフィックス・データをクリップする。さらに、適用されるシェーディング・アルゴリズムに基づいて、照明モデルは、種々の箇所で評価される(例えば、プリミティブの頂点、および/または、プリミティブによってカバーされた画素)。次に、変換されクリップされたグラフィックス・データは、ラスタライゼーション・ステージ110dに送られる。このラスタライゼーション・ステージでは、変換されたプリミティブを画素に変換し、一般に各プリミティブのコントリビューション(contribution)を各画素に格納する。レンダリング・エンジン110cは、種々のアーキテクチャで構成することができる。このようなアーキテクチャのより詳細な説明は、刊行物“Computer Graphics: Principles and Practice”, pp. 855-920 (2nd Ed. 1990), Foley等著に見い出すことができる。本実施例では、レンダリング・エンジン110cは、マルチスレッド並列プロセッサとして実現される。

【0024】従来のように、フレームバッファ110fは、図3の表示装置112の各画素に対してカラーを与える画素データを格納する。画素データは、フレームバ

バッファ110fから周期的に出力されて、表示装置112に表示される。好ましくは、フレームバッファ110fは、それぞれがnビット深さの行列マトリックスとして配列される。特定の行列アドレスは、典型的に、表示装置112の表示領域内の画素位置に相当している。例えば、(0, 1)の(行, 列)アドレスは、表示装置112の位置(0, 1)での画素に相当している。各行は、典型的に、表示装置112の特定の走査ラインの画素を示し、各列は、典型的に、表示装置112の垂直ラインに沿って配列された画素を示している。各画素アドレスでのnビットは、画素に関する情報をコード化する。例えば、Zバッファ110eの各画素アドレスに格納されたnビットは、その画素において見えるオブジェクトの深さを示している。

【0025】グラフィックス・サブシステム110は、2つのフレームバッファを有することができる。一方のフレームバッファは、アクティブ表示部として働き、他方のフレームバッファは、次の表示のために更新される。両方のフレームバッファ共に、システム100の必要性に従って、アクティブからインアクティブで変化しうるが、転換(changeover)が達成される特定の方法は、本発明には関係しない。

【0026】さらに、フレームバッファ110fの構成が、表示装置112の表示領域に対応しないならば、スケーリング操作を、フレームバッファ110fに格納された画素値に対して実行することができ、これによりフレームバッファに格納されたイメージを縮小または拡大する。画素カラー値を複写することによって、またはカラー値間で一次または双一次補間を行って、フレームバッファ110fに格納された元の画素値間のギャップを埋めることによって、スケールアップを得ることができる。隣接する画素のカラー値を平均化することによって、スケールダウンを得ることができる。

【0027】図4をさらに詳細に説明すると、共通のグラフィックス・サブシステム110は、グラフィックス・サブシステム110の動作を指示するグラフィックス制御プロセッサ110bを有している。このグラフィックス制御プロセッサ110bは、制御バス110gを介して、グラフィックス・サブシステム110の他の要素によって実行される動作を制御する。グラフィックス・サブシステム110は、バスインターフェース110aを介して、システムバス106に接続されている。バスインターフェース110aは、システムバス106の通信プロトコルに従って、システムバス106からデータを読み取り、システムバス106へデータを書き込む。

【0028】グラフィックス・サブシステム110は、ジオメトリ・サブシステム110cと、バスインターフェース110aに接続されたラスタライザ110dとを有している。ラスタライザ110dは、Zバッファ110eとフレームバッファ110fとに接続されている。ジ

オメトリ・サブシステム110cは、グラフィックス・データについて変換操作およびクリッピング操作を実行する。特に、ジオメトリ・サブシステム110cは、必要ならば、グラフィックス・データを、システムメモリ104に格納されたモデルの固有座標系からワールド座標系に変換する。これは、複数のモーリング変換行列の連結である単一変換行列で、各プリミティブの頂点を変換することによって、行うことができる。さらに、各プリミティブまたは頂点に関係した1つ以上の面法線ベクトルが、(適用されるシェーディング方法に基づいて)変換される必要がある。

【0029】ジオメトリ・サブシステム110cは、各プリミティブにビュー変換を行うこともできる。ビュー変換は、プリミティブの座標系を、ワールド座標系からビュー座標系に変換する。ビュー座標系の原点は、好適には、ビューウィンドウの中心にある。グラフィックス・データが、三角形の頂点となる場合には、ビュー変換操作により、ビュー座標系における三角形の頂点のリストが得られる。さらに、ジオメトリ・サブシステム110cは、また好適には、各プリミティブのビュー座標上に透視投影を実行して、透視縮小を与える。ジオメトリ・サブシステム110cの変換操作のより詳細な説明は、刊行物Computer Graphics Principles and Practice, pp. 201-281, 866-869 (2nd Ed. 1990), Foley, Van Dam, FeinerおよびHughes著に見出すことができる。

【0030】ジオメトリ・サブシステム110cは、また、クリッピング操作を実行することができる。このクリッピング操作では、プリミティブが、クリッピング・ボリュームに対してクリップされて、可視の変換されたプリミティブの部分を定める。さらに、ジオメトリ・サブシステム110cは、クリッピング操作により出力されたプリミティブの頂点の座標を、ラスタライザ110dによって要求される正規化装置座標系にマップする。レンダリング・パイプラインにおけるこのステップの結果は、プリミティブの可視の部分を記述する正規化装置座標系における頂点のリストである。クリッピング操作のより詳細な説明は、刊行物Computer Graphics Principles and Practice, pp. 110-132, 924-945, 869-870 (2nd Ed. 1990), Foley, Van Dam, FeinerおよびHughes著に見出すことができる。

【0031】さらに、グラフィックス・サブシステム110は、3次元ビュー/モデルのオブジェクトの表面上の光源の影響をシミュレートするライティング(lighting)計算を行うことができる。典型的に、ライティング計算は、(a) ビューアーの特性、(b) レンダリングされるオブジェクトの特性、(c) 1つ以上の

光源の特性に基づいている。ビューアーの特性は、レンダリングされるオブジェクトに対するビューアーの位置を含むことができる。オブジェクトの特性は対象物体を定める三角形の各頂点の位置および法線ベクトルを含むことができる。光源の特性は、種類（周囲光、指向性光、スポットライトなど）に依存し、強度、カラー、方向性、減衰率、テーパ角度を含むことができる。このようなライティング計算を行う方法のより詳細な説明は、刊行物 *Computer Graphics Principles and Practice*, pp. 721-814 (2nd Ed. 1990), Foley, Van Dam, Feiner および Hughes 著に見出すことができる。

【0032】典型的には、ライティング計算は、ビューのオブジェクトの三角形のすべての頂点についてのレンダリング・プロセスの際に、1回実行される。したがって、ライティング計算を、ジオメトリ・サブシステム 110c によって行うことができる。しかも、ライティング計算は、すべての画素に対して行うことができる。典型的には、これは、ラスタライザ 110d によって実行されるシェーディング計算と共に実現される。この場合には、ライティング計算は、ラスタライザ 110d によって行われるシェーディング計算に含められる。

【0033】ラスタライザ 110d の動作を、3つのタスク、すなわち走査変換、シェーディング、ビジビリティ (visibility) 決定に分割することができる。走査変換は、プリミティブの可視部を、個々の画素に分解する。シェーディングは、各画素のカラーを演算する。ビジビリティ決定は、各画素でのプリミティブの Z 軸（または深さ値）を用いて、プリミティブについて “ビジブル (visible)” である画素の組を演算する。したがって、プリミティブの可視部によって覆われる各画素について、ラスタライザ 110d は、画素情報、例えばプリミティブのカラーと深さを生成し、適切なときに、画素でのプリミティブのカラー情報および深さをフレームバッファ 110f と Z バッファ 110e にそれぞれ書き込む。ラスタライザ 110d の動作のより詳細な説明は、刊行物 *Computer Graphics Principles and Practice*, pp. 649-720, 870-871 (2nd Ed. 1990), Foley, Van Dam, Feiner および Hughes 著および米国特許第 4,805,116 号明細書に見出すことができる。これら文献の内容は、本願明細書の内容として引用される。

【0034】適切なグラフィックス・レンダリング・エンジンのすべてのアーキテクチャについて説明したが、グラフィックス・レンダリングの單一スレッド内に 3 つの主要な要素があることに留意すべきである。以下、これら要素を、状態管理、ジオメトリ計算、ラスタライゼーションと言うものとする。

【0035】状態管理機能は、ライン幅、光ポジション、ビューアー・ポジションなどのようなレンダリング・パラメータを変更する。状態管理機能は、ジオメトリ計算、ラスタライゼーション計算、あるいはこれらの両方に影響を与えるものとして、明瞭に区別すべきである。状態変数は、適切なプロセッサに格納される。例えば、ジオメトリ計算に関連した状態は、ジオメトリ計算を実行するプロセッサに格納される。

【0036】ジオメトリ計算機能への入力は、モデリング座標において特定されたデータを有する 1 組の頂点である。ジオメトリ計算は、モデリング座標から正規化装置座標 (NDC) への座標変換、クリッピング、ライティング、テクスチャ、フォグ (fog) 評価を含んでいる。最終結果は、ラスタライザが支持するプリミティブに組立てられることが必要な 1 組の頂点である。

【0037】ラスタライゼーション機能は、NDC 空間ににおいて定められるプリミティブのフラグメントへの変換と、フラグメント上での画素操作の実行、宛先バッファ（フレームバッファ自体または画素マップ）の更新を含んでいる。

【0038】必要なグラフィックス作業の区分化に関して、多くの状況において、アプリケーションは、ただ 1 つのグラフィックス・コンテキストおよびただ 1 つのスレッドとを有することに留意すべきである。このようなアプリケーションを並列化するためには、グラフィックス・アクセラレータによって採用されている従来の手法を適用することができる。

【0039】グラフィックス・データ・ストリームは、本質的に直列の性質を有している。したがって、従来の並列グラフィックス・アーキテクチャは、並列演算のパイプライン・モデルを用いている。これは、スーパーパイプライン化プロセッサにおける命令レベルの並列化に類似している。この手法では、グラフィックス・パイプラインは、十分に区分された操作（すなわち、状態管理、ジオメトリ計算、ラスタライゼーション）の上記シーケンスに組み入れられ、異なるプロセッサで実行される。

【0040】この手法は、一般に満足すべきものであるが、かなりの量の全処理時間が、1 つのプロセッサから他のプロセッサへのデータ移動において費やされる。というのは、プロセッサが、アドレス空間を共用しないからである。したがって、かなり最近の研究は、パイプライン化に加えて、データ並列化を用いることに集中している。このことは、複数の実行ユニットに対してデータ並列化を利用するスーパースカラー・プロセッサで採用されている手法に類似している。PowerPC 604 に完了バッファを使用するような方法は、順序のはずれた実行および以降の同期を許容するように行われてきた。これは、グラフィックス・プリミティブが、アプリケーションによって特定された順序でレンダリングされ

なければならないという事実に大部分よっている。しかし、ラスタライゼーションに対して、ウィンドウ空間における相互実行を用いるデータ並列化を利用する他の方法がある。

【0041】全ての状態管理機能は、本来的に順次的である。さらに、各状態管理機能は、間接的に同期ステップを含んでいる。さらに、状態管理機能は、状態変数および機能ポインタを典型的に変更し、したがって演算的に強力ではない。状態管理機能を並列化することによって、多くのことが得られることは明らかではないが、大部分の状態管理機能を連続して実行することが好ましい。行列およびマテリアル (material) の変更のケースは、よりわずかな注意で並列に処理される。これは、行列状態を、頂点バッファのための状態にコピーすることによって行うことができる。

【0042】ジオメトリ計算機能に関しては、Open GLインタフェースのような多くの一般的なグラフィックス・インタフェースは、むしろ微細粒度であり、少量のジオメトリ・データを、アプリケーションからレンダリング・コードに送る（例えば、一度に頂点を）。したがって、頂点レベルでのデータ並列化は、データがアプリケーションから受取られる場合には、実際的でない。その結果、有用な並列化を利用する前に、入力ジオメトリ・データをバッファすることが典型的に必要である。バッファリング・プロセス自体は、本来的に順次的である。

【0043】理論的には、アプリケーションからのいくつかの頂点についてのデータが、バッファされた後に、プロセッサ間に配布することができ、変換、ライティング計算、テクスチャ計算、フォグ計算を、各頂点に対して並列に行うことができる。しかし、この手法は、並列化の粒度が非常に低いので、劣った性能を生じることがわかった。

【0044】したがって、本発明では、バッファリング・スレッドは、現在のカラー、テクスチャ、面法線座標を、必要に応じて、頂点データ構造にコピーし、頂点をバッファに格納するのが好適である。バッファリング・スレッドは、また、頂点に関係したデータ以外の状態管理コールがあったときに、頂点バッファをフラッシュする。バッファリング・スレッドは、また、このバッファを、コンテキスト（またはアプリケーションレベル・スレッド）についてのバッファの待ち行列に加える。他のスレッドまたはスレッドの組を用いて、待ち行列からバッファを獲得し、バッファをラスタライザに渡す前に、ライティング、テクスチャリング、NDC座標への変換などのような作業の残りを完了する。

【0045】図5は、全体のプロセスを示す。図5では、グラフィックス・アプリケーション10は、関連したグラフィックス・ライブラリ・クライアント12と、グラフィックス・ライブラリ・サーバ14とを有してい

る。頂点バッファ16、18、…、Nのような入力グラフィックス作業バッファは、1個の主スレッドおよびN個の従スレッドへサブタスクを与える。従スレッドの各々は、関連するジオメトリ・ライブライン20、22、…、Nとを有している。出力バッファ24、26、…、Nは、ジオメトリ計算の結果を、ラスタライザ・インターフェース28に、次にラスタライザ・ハードウェア30に供給する。一般に、ブロック10～14は、図4のグラフィックス制御プロセッサ110bに相当し、ブロック16～26は、図4のジオメトリ・サブシステム100cに相当し、ブロック28、30は、図4のラスタライザ110dに相当している。

【0046】図5はまた、変数フラグを格納するローカル変数セット32を示している。変数フラグは、以下に詳細に説明するように、主スレッドまたは従スレッドによって、セット、リセット、テストできる。ローカル変数セット32は、対応するプロセッサ（図1参照）の各々に関連したキャッシュメモリに格納することができ、この場合には、キャッシュ・コヒーレンシィまたは他の適切な機構を用いて、主スレッドおよび従スレッドが、常に、同じ変数状態へのアクセスを有することを保証する。

【0047】本発明の教示は、固有のロードバランシング方式の使用により大きい性能利得を得るために、ジオメトリ・ライブラインをレンダリングするソフトウェア・グラフィックス（2次元または3次元）を並列化する方法を提供する。ロードバランシング方式は、最適数の頂点バッファと、ライブライン直列処理ステップを、並列処理ステップに区分して分離する方法を用いている。個々のライブライン並列処理ステップは、自身のカウンタおよびフラグを有し、各ユーザ・スレッドは、また、自身のカウンタ、フラグ、最適数のフラッシュ・バッファを有し、時間を消費するスレッド同期機能の使用を排除する。また、アプリケーションに関係するスレッドにのみ可視である変数を用いることによって、スレッド・ライブラリ内のロック／アンロック機能に頼ることなく、2つのスレッド内で作業バッファを共用する方法を提供する。

【0048】並列データプロセッサ上でタスクを共用する場合、共働プロセスが、有限の時間内で、タスクを処理することを保証し、他方ではタスクが1つのプロセスで1回だけ実行されることを保証することが必要になる。実施例では、2つのプロセス、すなわち主プロセスおよび従プロセスが用いられて、前述したジオメトリ計算タスクのうちの1つのようなタスクを実行する。所定の（有限）時間内で、タスクが1回実行される限り、どのプロセスがタスクを実際に実行するかは重要ではない。

【0049】本発明の実施例では、以下の変数（表1）が用いられる。mvbは、主スレッドによって扱われる

現行のバッファのインデックスであり、cvbは、従スレッドによって扱われる現行のバッファのインデックスであり、mflag[b]およびcflag[b]は、同期に用いられる変数であり、4つの可能な値、すなわちエンティティ(empty), フル(full), 主(main), 従(child)をとりうる。主スレッドは、mflag[b]およびcflag[b]の値を、emptyから、mainまたはchildに変えることができ(表2参照)また、mflag[b]の値を、emptyからprocessedに変えることができる。従スレッドは、mflag[b]の値を、processedからemptyに、cflag[b]の値を、childからemptyに変えることができる。変数owner[b]は、mainまたはchild

dのいずれかのスレッドが、バッファbを処理しているかを示すために用いられる。変数owner[b]およびstate[b]は、主スレッドによってのみ、割当ることができる。変数state[b]は、バッファの最大サイズに達する故に、またはアプリケーションによって処理される現行のジオメトリ・プリミティブを終了させるEnd機能コールを受取ることによって、バッファがいかに満たされたを示すために用いられる。フラグを含むこれら変数は、ローカル変数セット32に好適に格納される。ローカル変数セットでは、変数は、オペレーティング・システム・コールを行う必要なしに、主スレッドおよび従スレッドにアクセスできる。

【0050】

【表1】

| 変数 | 主スレッド | 従スレッド |
|-----------|---------------------------|---------------------------|
| 頂点バッファ# | mvb | cvb |
| 頂点バッファフラグ | mflag[mvb], cflag[mvb] | cflag[cvb], mflag[cvb] |
| オーナー | owner[mvb] | owner[cvb] |
| 状態 | state[mvb] | state[cvb] |

【0051】以下の操作は、変数上で主スレッドおよび従スレッドによって行うことができる。

【0052】

【表2】

| スレッド | mflag | cflag | owner | state |
|------|-----------|-------|-------|-------|
| 主 | main | main | main | end |
| | child | child | child | full |
| | processed | | | |
| 従 | empty | empty | | |

【0053】図6, 7, 8および図9, 10のフローチャートは、本発明の教示に従って、主スレッドおよび従スレッドの操作をそれぞれ示しており、ジオメトリ・パイプラインの並列化における制御フローを示している。本発明の好適な実施例では、従スレッドは、処理された頂点バッファを、ラスタライザ110dにまで送る。主スレッドまたは従スレッドは、頂点バッファを処理することができることに留意すべきである。

【0054】次に、図6～図8の主スレッドのフローチャートを説明する。

【0055】図6において、必要なデータ構造を初期化することによって、主スレッドが開始する(ステップ40～44)。次に、主スレッドは、非状態コマンドが到着するまで、状態コマンドを受取り始める(ステップ46)。このコンテキストにおける状態コマンドは、ジオメトリック処理操作の全状態の記述であり、ポリゴンの種類、カラー、ライティング方向などを含むことができる。すなわち、状態コマンドは、頂点データを正しく処理するために、主および従のスレッドによって必要とされる情報を含む。非状態コマンドは、処理されるべき頂

点データ(ジオメトリック入力データ)を含むことができる。状態コマンドは、主スレッドおよび従スレッドによってアクセスできるメモリ領域に格納され、他方、頂点データは、1つ以上の頂点データ入力バッファ16, 18に格納される。

【0056】従スレッドが生成されていなければ(ステップ48)、従スレッドが生成される(ステップ50)。次に、従スレッドは、図9、図10の従スレッドフローチャート(以下に説明する)に記載されているように、実行を開始する。次に、主スレッドは、ジオメトリック入力データの受取りを開始し(ステップ52)、頂点バッファ16への格納を開始する(ステップ54, 56, 58, 60)。頂点バッファが、フルになるか、あるいはアプリケーションによる格納が終了すると、頂点バッファは、以下に詳細に説明するように、主スレッドまたは従スレッドによって処理される(ステップ62, 64, 66, 68)。

【0057】前記の非状態コマンドが到着し始めると、従スレッドが作成されていると(ステップ48)、制御は、ステップ70, 72に進む。ステップ70, 7

2では、最後の頂点バッファが処理されたので、新しいコマンドが到着しているか否かの判別がなされる。新しい状態コマンドの到着は、新しいパラメータ（例えば、カラー、ピューポイント、およびまたはライティング方向は変わり得る）を用いて、ジオメトリック処理が行われることを意味している。このようなコマンドが受取られていなければ、主スレッドは、ステップ52に戻り、ジオメトリック入力データを受取る。しかし、未処理状態コマンドが存在するならば、主スレッドが代りにすべての頂点バッファをループし、それらのすべてがエンプティになるまで待つ（ステップ74, 76, 78, 80）。このことは、事前に存在する頂点データの処理が、前の状態コマンドに含まれるパラメータを用いることによって、行われることを可能にする。すべての頂点バッファが、エンプティになると、主スレッドは、ステップ80からステップ52に戻って、ジオメトリック入力データの受取りを再び開始する。

【0058】入力バッファが主スレッドに割当てられると、すなわちステップ62で`mf lag [mb] == main`ならば、主スレッドは、図7に示すように、バッファを処理し（ステップ82～98、図7のステップ98と、図6のステップ66とは、同じステップである）、次に、制御は図8のステップ100に進む。制御はまた、バッファが從スレッドに割当てられた場合（ステップ62, 68）、ステップ100に進む。図8において、主スレッドは、次の物理的頂点バッファにどのバッファが割当てられるかを決定する。ステップ100では、バッファ・カウンタ`mb`は、最初にインクリメントされる。從スレッドが、配布の最終ラウンドからの“次の次（`next to next`）”の物理的頂点バッファに対してビジーでないならば、主スレッドは、從スレッドの処理が普通に行われ、從スレッドが、処理（ステップ102）に対するバッファのバックログを有さないことを知っている。この場合、主スレッドは、次の物理的頂点バッファを從スレッドに割当てる（ステップ104）。しかし、從スレッドが前記バッファ（次の次の物理的頂点バッファ）を依然として処理するならば、主スレッドは、從スレッドが`c flag [mb] == empty`（ステップ106）であるか否かをチェックすることによって、從スレッドが`mb`番目のバッファをエンプティにしたか否かを判別する。從スレッドが、このバッファをエンプティにしなかったならば、少なくとも1個のバッファがフリー（エンプティ）となるまで、主スレッドは遅延する。というのは、この状況では、新しいジオメトリック入力データを受取るエンプティバッファが存在しないからである。從スレッドの処理がキャッチアップするまで、主スレッドは、ループに留まる（ステップ102, 104）。主スレッドが、從スレッドがバッファのかなりのバックログを有すること、すなわち`c flag [mb+1]`がエンプティでなく、

`c flag [mb]`がエンプティであることを判定するならば、それは次のバッファを主スレッドに割当て（ステップ108）、次のバッファを初期化し（ステップ110）、モード変数および状態変数を更新する（ステップ112）。次に、主スレッドはステップ46（図6）に戻り、現行のモードおよび状態を入力し、状態情報およびジオメトリック入力データを受取る主スレッドによって処理が続く（ステップ52）。

【0059】次に、図9および図10の從スレッドのフローチャートを説明する。

【0060】從スレッドが生成された後（図6のステップ50）、データ構造および從バッファの初期化（ステップ120, 122, 124）を含む初期化処理を実行する。ステップ126では、從スレッドを続けることができるか否かを判別する。スレッドを続けることができないならば、制御は、ステップ128に進み、このステップでは、從スレッドが終了する。從スレッドを続けることができるならば、制御は、制御はステップ130に進む。このステップでは、從スレッドが、從スレッドに割当てられた入力バッファが存在するか否か、すなわち`c flag [cb] == chidl`であるか否かを連続してチェックする。

【0061】このようなバッファが存在するならば、制御は、Aを経て図10に進む。図10では、バッファ処理状態に入る前に、從バッファが初期化される（ステップ132）。まず最初に、ステップ134で、`state [cb-1] == end`であるか否か、すなわち前のバッファが終了した環境（例えば、`GLEND`の際に、前のバッファを終了させた）をチェックする。Yesならば、制御はステップ142に進み、Noならば、制御はステップ136に進む。ステップ136では、從スレッドは、どのスレッドが前のバッファを処理したか、すなわち前のバッファは、主スレッドによって、または從スレッドによって所有されたかを判別する。ステップ138, 140では、從スレッドは、前の物理的頂点バッファからのデータをコピーする。これは、このようなコピーの必要性がある場合である。ステップ142では、從スレッドは、現行の主バッファのデータをコピーし、ステップ144では、`state [cb] == end`であるか否かをチェックする。Noならば、バッファはフルであるとみなされて、ステップ146で処理され、Yesならば、頂点バッファは、エンド（最終）バッファであるとして認識され、完全にフルとはなり得ない。この場合における処理は、ステップ148で行われる。バッファが処理された後に、ラスタライザ・インタフェース28を介して、バッファは、從スレッドによって、ラスタライザ・ハードウェア110dにフラッシュされる（ステップ150）。次に、從スレッドは、`c flag [cb] = empty`と設定することによって、バッファをエンプティにし、カウンタ`cb`をイン

クリメントする（ステップ160）ことを示している。制御は、Bを経て図9のステップ126に戻る。

【0062】図9のステップ130に戻り、従スレッドに割当てられた入力バッファが存在しないならば、従スレッドは、ステップ162で、主スレッドによって処理されたバッファがシーケンスに存在するか否かを判別する。Noならば、従スレッドは、ステップ130と162との間をループする。これは、頂点バッファがプロセスに割当てられるか、あるいは頂点バッファが主スレッドに割当てられるまで行われる。ステップ162で、主スレッドがバッファに割当てられたことが判別されると、従スレッドは、バッファがラスタライザ・ハードウェア110dに送られる（フラッシュされる）準備ができるか否かを判別する。バッファがラスタライザ110dにフラッシュされると、バッファはエンプティであるとマークされる。ステップ164では、従スレッドは、主スレッド・バッファが、エンプティ（すでにフラッシュされた）であるとマークされるか否かを判別する。Noならば、制御はステップ166に進む。ステップ166では、従スレッドは、主スレッドが、自己割当てされた頂点バッファの処理を終了したか否かをチェックする。Noならば、従スレッドは、主スレッドが、頂点バッファの処理を終了するまで、ステップ164と166との間をループする。ステップ166でYesならば、従スレッドは、ステップ168でバッファをフラッシュし、ステップ170でバッファをエンプティとしてマークする。次に、ステップ164で、エンプティ状態が検出され、制御は、ステップ172および174に進む。ステップ172および174では、従バッファは、エンプティとマークされ、バッファ・ポインタはインクリメントされる。次に、制御は、ステップ126に戻る。

【0063】この方法を、2つ以上の従スレッドに拡張するためには、`cflag`のような1組のスレッド変数を生成する。この状況では、従スレッドの1つは、主/従スレッドとして機能し、主/従スレッドのみが、処理された頂点バッファを、ラスタライザ110dに送ることができる。

【0064】システムに用いられる頂点バッファの数は、システムを効果的にするために、スレッドの数よりも大きくするのが好適である。そうでなければ、いくつかの従スレッドは、作業バッファを処理しようとして、作業バッファのいくつかは、アイドルであることが要求され、したがってシステム・リソースをむだにする。好ましくは、ユーザ・スレッド（主なわち初期のスレッドを含む）の数は、マルチプロセッサ・システムのマイクロプロセッサの数以下であり、各マイクロプロセッサへのロードは、比較的バランスしている。

【0065】従スレッドのみが、処理バッファをフラッシュすることのできる並列処理システムについて説明し

たが、他の実施例では、スレッドが処理頂点バッファをフラッシュすることを可能にすることが望まれる。しかし、発明者は、従スレッドのみが処理バッファをフラッシュするときに、リソースの最大効率および使用が得られるものと判断した。

【0066】前述した方法によれば、スレッドの明確な同期が必要とされることがわかる。というのは、複数のスレッドの動作を同期する方法は、アプリケーションに関係するスレッドに対して可視である変数を採用しているからである。すなわち、同期変数は、図5のローカル変数セット32において、主スレッドおよび従スレッドに対して、ローカルに得られるので、明確なオペレーティング・システム・コールなどは要求されない。

【0067】本発明は、さらに、グラフィックス処理パイプラインの3つの直列処理ステップが、4つの並列処理ステップに区分されて分離され、各パイプライン並列処理ステップが、自身のカウンタとフラグとを有し、各ユーザ・スレッドも自身のカウンタと、フラグと、最適数のフラッシュ・バッファとを有する方法を教示している。

【0068】頂点バッファの最適数は、頂点バッファの最小数以下である。このことは、各ユーザ・スレッドに、十分な数の頂点バッファが、多くのパイプライン応用の場合に、大半の時間ビジーとなることを許容する。各スレッドに対するフラッシュ・バッファの最適数は、さらに、共用頂点バッファの最適数以上である。頂点バッファは、主なわち初期スレッドによって、ファーストイン/ファーストアウトで順次に格納されおよびフラッシュされる。好ましくは、主なわち初期スレッドと、従なわちユーザ・スレッドは、それらの相互に排他的なフラッシュ・バッファを順次に格納する。

【0069】このようにして、本発明の教示によって解決された問題は、並列処理の領域では基本的であり、および本発明の教示は広い適用可能性を要することを理解すべきである。このように、本発明の教示は、前述したグラフィックス処理応用のみに制限されるものと解釈すべきではない。

【0070】本発明を、特に、好適な実施例に基づいて説明したが、本発明の範囲と趣旨から逸脱することなく、形態および詳細の変形を行うことができることは、当業者には理解できるであろう。

【0071】まとめとして、本発明の構成に関して以下の事項を開示する。

(1) マルチプロセッサ・システムにおいて、入力データをデータプロセッサ・パイプラインで処理する方法であって、主スレッドを操作して、入力バッファがフルになるまで、あるいは入力データが終了するまで、前記入力バッファに前記入力データを格納するステップと、従スレッドが存在しない場合には、従スレッドを生成して、前記入力バッファを、処理のために従スレッドに割

当てるステップと、従スレッドが既に存在する場合は、前記従スレッドを前記入力バッファに割当てることができるか否かを判別し、割当てることができる場合は、前記入力バッファを、処理のために従スレッドに割当て、割当てることができない場合には、前記入力バッファを、前記従スレッドによって実行される処理との並列処理のために、前記主スレッドに割当てるステップとを含み、前記割当てるステップおよび判別するステップは、前記主スレッドおよび従スレッドの両方にアクセス可能なローカル変数を用いる、ことを特徴とする方法。

(2) 処理データバッファを前記パイプラインの連続する処理ステージに送るステップをさらに含み、このステップは、前記データバッファが、前記従スレッドによってまたは前記主スレッドによって処理されたかにかかわらず、前記従スレッドによってのみ実行されることを特徴とする上記(1)に記載の方法。

(3) 処理データバッファを前記パイプラインの連続する処理ステージに送るステップをさらに含み、このステップは、前記データバッファが、前記従スレッドによって、他の従スレッドによって、または前記主スレッドによって処理されたかにかかわらず、前記従スレッドによってのみ実行されることを特徴とする上記(1)に記載の方法。

(4) 前記入力データは、ジオメトリ・イメージをレンダリングするための、頂点座標の記述であることを特徴とする上記(1)に記載の方法。

(5) マルチプロセッサ・システムにおいて、主スレッドの操作と、少なくとも1つの従スレッドの操作とを同期させる方法であって、主スレッドを操作して、入力バッファがフルになるまで、あるいは入力データが終了するまで、前記入力バッファに前記入力データを格納するステップと、従スレッドが存在しない場合には、従スレッドを生成して、前記入力バッファを、処理のために従スレッドに割当てるステップと、従スレッドが既に存在する場合には、前記従スレッドを前記入力バッファに割当てることができるか否かを判別し、割当てることができる場合には、前記入力バッファを、処理のために従スレッドに割当て、割当てることができない場合には、前記入力バッファを、前記従スレッドによって実行される処理との並列処理のために、前記主スレッドに割当てるステップとを含み、前記割当てるステップおよび判別するステップは、オペレーティング・システム・コールの使用を要求することなく、前記主スレッドおよび従スレッドの両方にアクセス可能なローカル変数を用いて同期される、ことを特徴とする方法。

(6) 処理データバッファを前記パイプラインの連続する処理ステージに送るステップをさらに含み、このステップは、前記データバッファが、前記従スレッドによってまたは前記主スレッドによって処理されたかにかかわらず、前記ローカル変数を用いて、前記従スレッドによ

ってのみ実行されることを特徴とする上記(5)に記載の方法。

(7) 処理データバッファを前記パイプラインの連続する処理ステージに送るステップをさらに含み、このステップは、前記データバッファが、前記従スレッドによって、他の従スレッドによって、または前記主スレッドによって処理されたかにかかわらず、前記ローカル変数を用いて、前記従スレッドによってのみ実行されることを特徴とする上記(5)に記載の方法。

(8) 前記入力データは、ジオメトリ・イメージをレンダリングするための、頂点座標の記述であることを特徴とする上記(5)に記載の方法。

(9) グラフィックス・ジオメトリ・パイプラインにおいて、表示のためにレンダリングされるグラフィカル・モデルを記述する頂点座標のデータストリームを処理する方法であって、主スレッドを操作して、前記グラフィカル・モデルの第1の状態に関連した頂点座標データストリームの開始を検出するステップと、第1のデータ入力バッファがフルになるまで、あるいは前記入力頂点座標データストリームが終了するまで、前記第1のデータ入力バッファに前記頂点座標データストリームを格納するステップと、第1の従スレッドが存在するか否かを判別するステップと、前記第1の従スレッドが存在しない場合には、第1の従スレッドを生成して、前記第1のデータ入力バッファを、処理のために前記第1の従スレッドに割当てるステップと、前記第1の従スレッドが既に存在する場合には、前記第1の従スレッドを前記第1のデータ入力バッファに割当てることができるか否かを判別し、割当てることができる場合には、前記第1のデータ入力バッファを、処理のために前記第1の従スレッドに割当てるステップと、第2のデータ入力バッファがフルになるまで、または前記入力頂点座標データストリームが終了するまで、前記第2のデータ入力バッファに頂点座標データストリームをさらに格納するステップと、前記第1の従スレッドを前記第1のデータ入力バッファに割当てることができないことが判別される場合には、前記主スレッドを操作して、以下の(A)または(B)のステップを実行するステップを含み、(A)第2の従スレッドが存在するか否かを判別し、前記第2の従スレッドが存在しない場合には、第2の従スレッドを生成して、前記第2のデータ入力バッファを、処理のために前記第2の従スレッドに割当て、前記第2の従スレッドが既に存在する場合には、前記第2の従スレッドを前記第2のデータ入力バッファに割当てることができるか否かを判別し、割当てることができる場合には、前記第2のデータ入力バッファを、処理のために前記第2の従スレッドに割当て、(B)前記第1および第2の従スレッドの少なくとも1つによって実行される処理と並列に処理するために、前記第2のデータ入力バッファを前記主スレッドに割当て、前記割当てるステップおよび前記判別

のステップは、オペレーティング・システム・コールの使用を要求することなく、前記主スレッドと前記第1および第2の従スレッドとを両方にアクセス可能なローカル変数を用いて同期される、ことを特徴とする方法。

(10) 処理データバッファを、前記グラフィックス・ジオメトリ・パイプラインの連続するステージに送るステップをさらに含み、このステップは、前記データバッファが、前記第1の従スレッドによって、前記第2の従スレッドによって、または前記主スレッドによって処理されたかにかかわらず、前記ローカル変数を用いて、前記第1の従スレッドによってのみ実行されることを特徴とする上記(9)に記載の方法。

(11) 前記連続するステージは、ラスタライザ・ステージであることを特徴とする上記(10)に記載の方法。

(12) 前記主スレッドを操作して、前記グラフィカル・モデルの第2の状態の発生を検出するステップと、前記グラフィカル・モデルの第1の状態に関連した頂点座標データのすべての処理を終了するステップと、をさらに含むことを特徴とする上記(10)に記載の方法。

(13) 複数のデータプロセッサを備え、各データプロセッサは、主スレッドを実行し、少なくとも1つの第2のデータプロセッサは、従スレッドを実行する、グラフィックス・データ処理システムであって、前記主スレッドと前記少なくとも1つの従スレッドとの両方によってアクセス可能な1組のローカル変数を格納するメモリ手段と、前記主スレッドの制御のもとで、入力グラフィックス・データストリームを格納する複数の入力バッファと、前記ローカル変数に応答する前記主スレッドに関連した処理手段であって、入力バッファを、前記主スレッドと前記従スレッドとの間に割当て、前記入力バッファに格納されたデータを用いて、グラフィックス・データ演算タスクを並列に実行するために、前記主スレッドの操作と従スレッドの操作とを同期させる処理手段と、を備えることを特徴とするグラフィックス・データ処理システム。

(14) 前記メモリ手段は、前記第1のデータプロセッサに接続された第1のキャッシュメモリと、前記第2のデータプロセッサに接続された第2のキャッシュメモリとによりなり、前記第1および第2のキャッシュメモリの各々は、前記1組のローカル変数の同一コピーを格納することを特徴とする上記(13)に記載のグラフィックス・データ処理システム。

【図面の簡単な説明】

【図1】本発明の好適な実施例によって用いることのできる従来のマルチプロセッサ・システムの基本的なブロック図である。

【図2】タスクを現在実行されている複数のスレッドに区分する従来の方法を示す図である。

【図3】本発明を実施するのに適したグラフィックス処理システムのブロック図である。

【図4】図3のグラフィックス・サブシステム・ブロックを詳細に示す図である。

【図5】区分され、本発明の好適な実施例に従って実行されるグラフィックス・タスクを示す図である。

【図6】主スレッドおよび従スレッドの動作をそれぞれ説明する論理フロー図であり、ジオメトリ・パイプラインの並列化における制御のフローを示す図である。

【図7】主スレッドおよび従スレッドの動作をそれぞれ説明する論理フロー図であり、ジオメトリ・パイプラインの並列化における制御のフローを示す図である。

【図8】主スレッドおよび従スレッドの動作をそれぞれ説明する論理フロー図であり、ジオメトリ・パイプラインの並列化における制御のフローを示す図である。

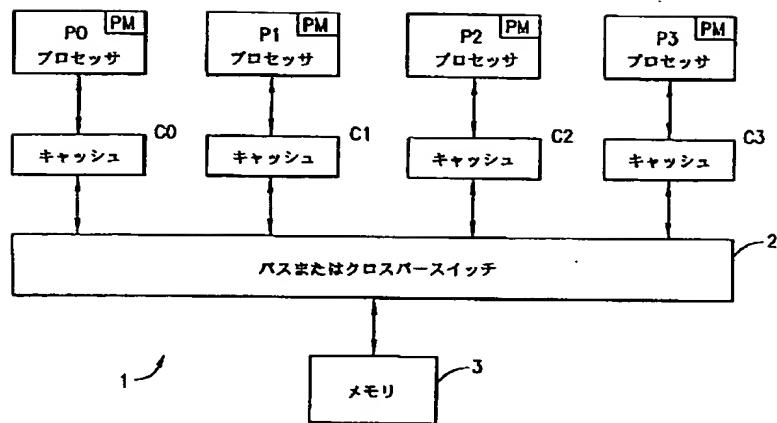
【図9】主スレッドおよび従スレッドの動作をそれぞれ説明する論理フロー図であり、ジオメトリ・パイプラインの並列化における制御のフローを示す図である。

【図10】主スレッドおよび従スレッドの動作をそれぞれ説明する論理フロー図であり、ジオメトリ・パイプラインの並列化における制御のフローを示す図である。

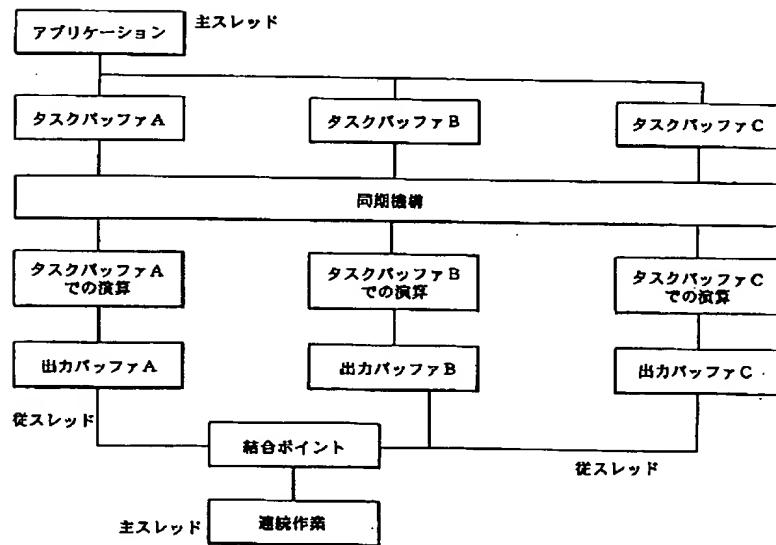
【符号の説明】

- 1 マルチプロセッサ
- 2 バスまたはクロスバースイッチ
- 3 システムメモリ
- 10 グラフィックス・アプリケーション
- 12 グラフィックス・ライブラリ・クライアント
- 14 グラフィックス・ライブラリ・サーバ
- 16, 18 頂点バッファ
- 20, 22 ジオメトリ・パイプライン
- 24, 26 出力バッファ
- 28 ラスタライザ・インターフェース
- 30 ラスタライザ・ハードウェア
- 32 ローカル変数セット
- 100 グラフィックス処理システム
- 102 システム制御プロセッサ
- 104 システムメモリ
- 106 システムバス
- 108 I/O装置
- 110 グラフィックス・サブシステム
- 110a バスインターフェース
- 110b グラフィックス制御プロセッサ
- 110c ジオメトリ・サブシステム
- 110d ラスタライザ
- 110e zバッファ
- 110f フレームバッファ
- 110g 制御バス
- 112 表示装置
- 114 アナログ-デジタル変換器

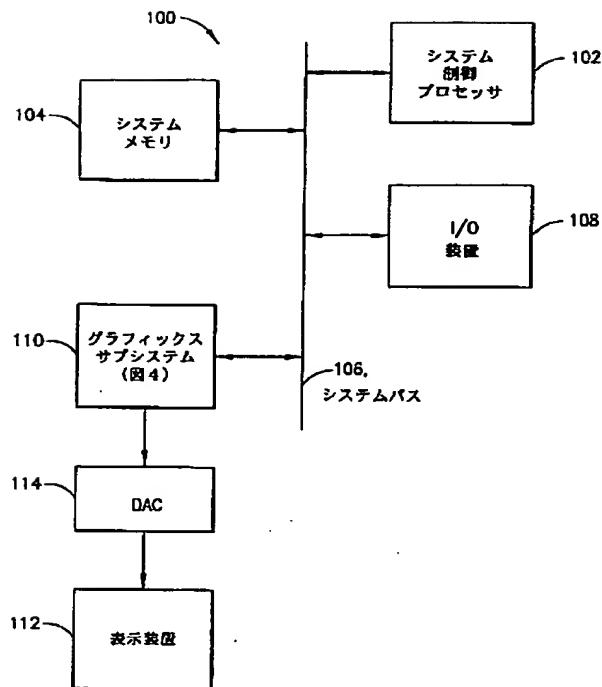
【図1】



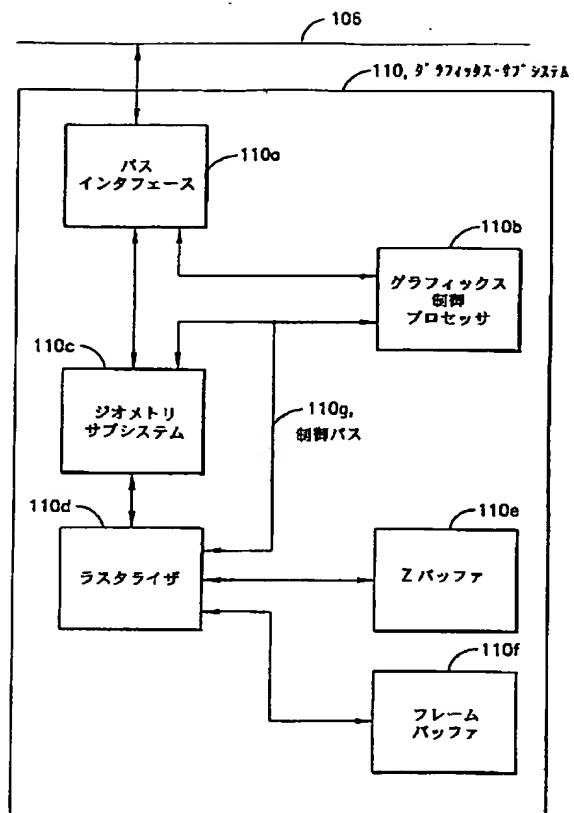
【図2】



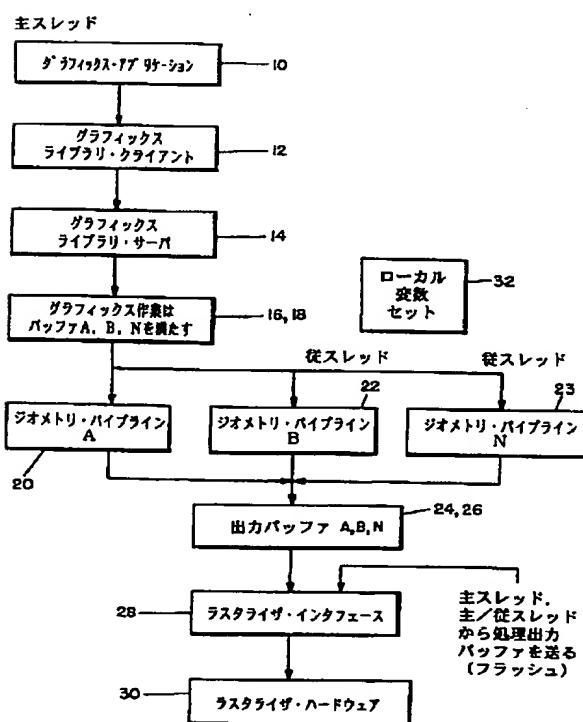
【図3】



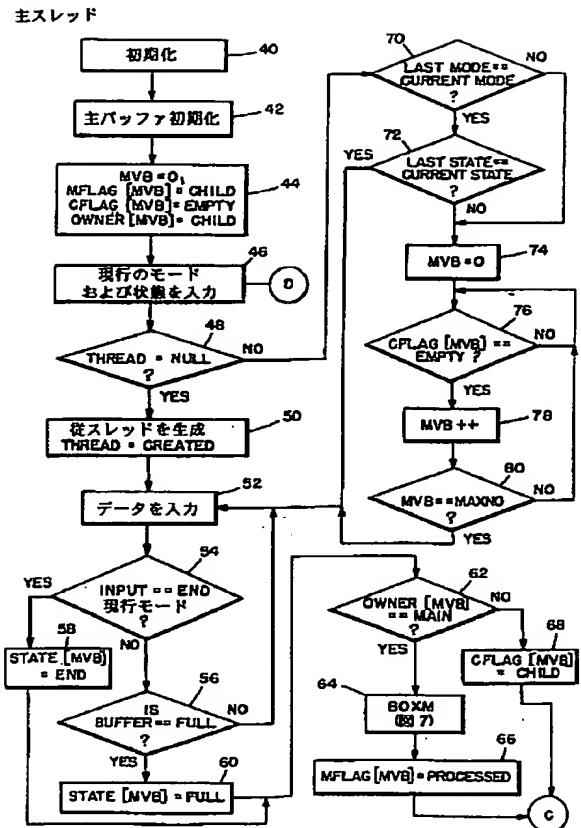
【図4】



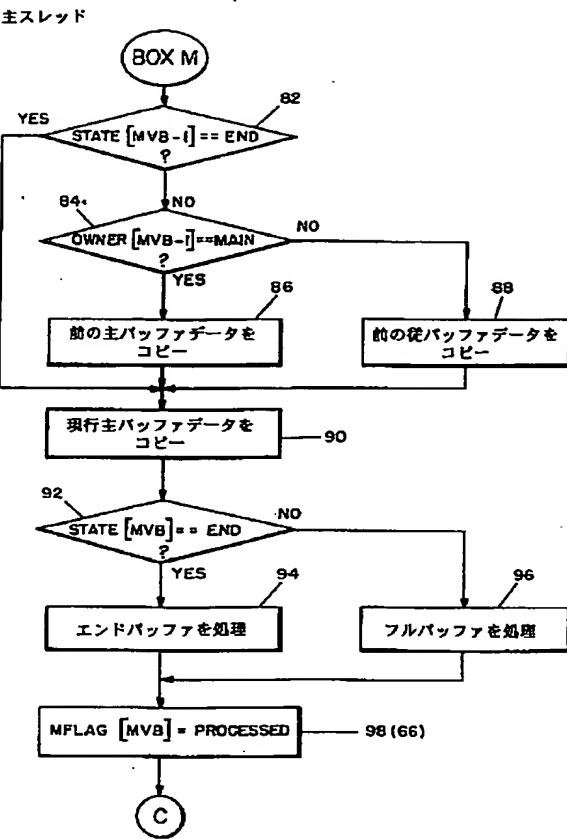
【図5】



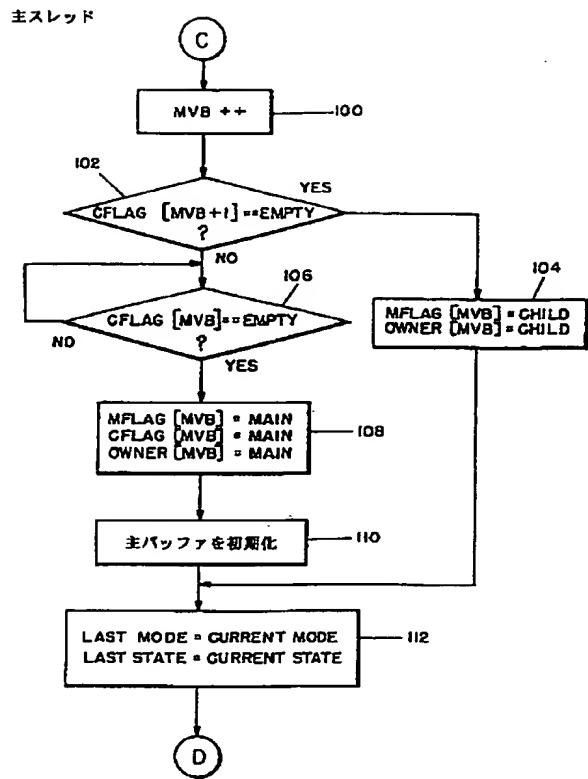
【図6】



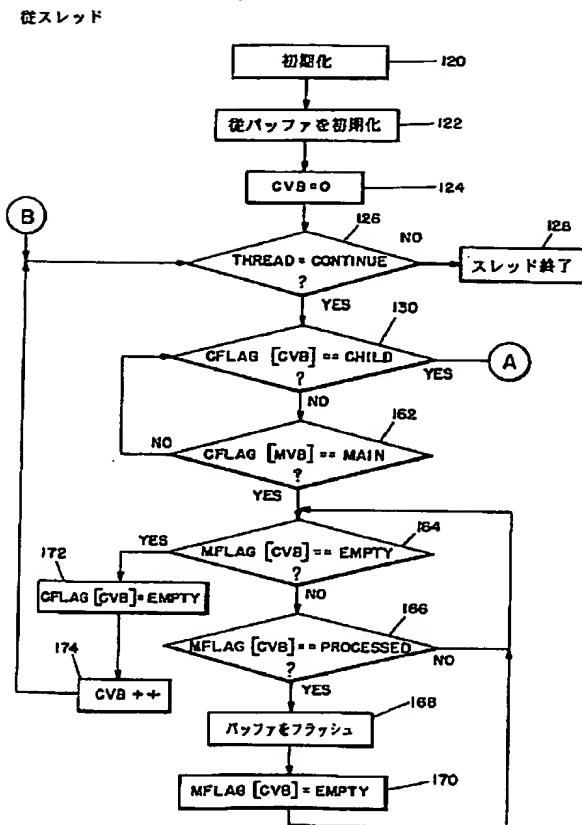
【図7】



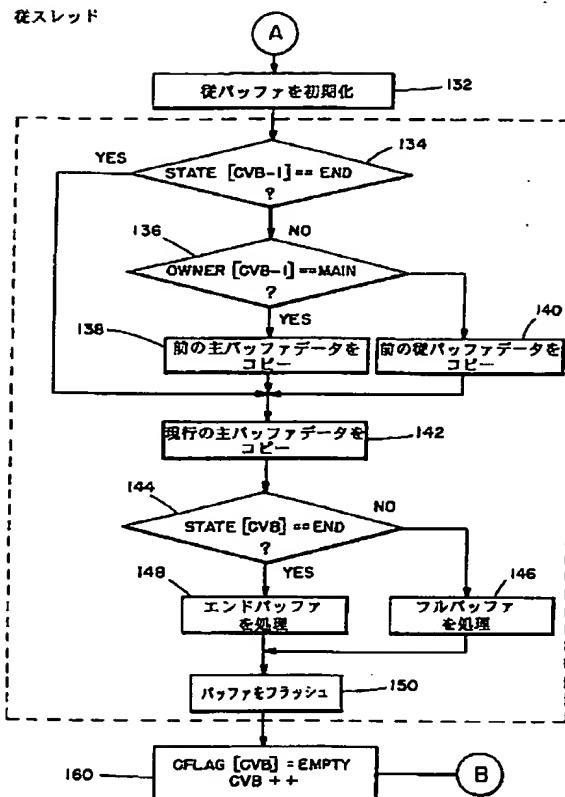
【図8】



【図9】



【図10】



フロントページの続き

(72)発明者 トーマス・ユーキュウ・クウォック
 アメリカ合衆国 07675 ニュージャージ
 州 ワシントン タウンシップ ビーチ
 ストリート 735

(72)発明者 チャンドラセクハール・ナラヤナスワミ
 アメリカ合衆国 06897 コネティカット
 州 ウィルトン ロング メドウス ロー
 ド 41
 (72)発明者 ベントーオレイフ・シュナイダー
 アメリカ合衆国 10598 ニューヨーク州
 ヨークタウン ハイツ クラウン ハイ
 ツ ロード 515